Universidad Andrés Bello

Estructura de Datos

CONJUNTOS

Universidad Andrés Bello
Facultad de Ingeniería

Contenido

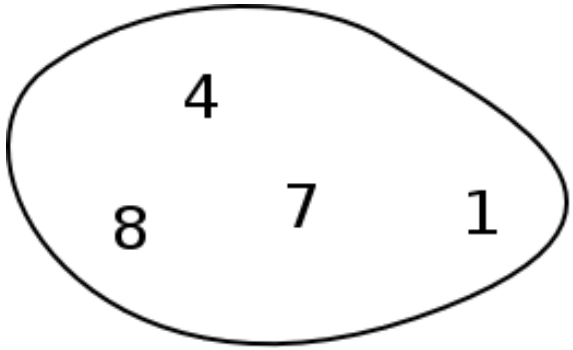
Introducción

- Los conjuntos son una de las estructuras básicas de las matemáticas, y por tanto de la ciencias de la computación.
- Conjunto está formado por una colección de elementos junto con sus operaciones, tal que:
  - Los elementos pueden ser datos simples o compuestos.
  - En general suelen ser todos los elementos del mismo tipo.
  - Pueden existir elementos repetidos.
  - Podría existir alguna relación de orden (lo que no implica que los elementos se encuentren ordenados).

Universidad Andrés Bello

Introducción

- **Universo de valores o conjunto universal**: es el conjunto de todos los posibles valores que puede haber en un conjunto.
- **Cardinal de un conjunto**: número de elementos que contiene.
- **Conjunto vacío** ($\{\}$): conjunto con cardinal cero ($\emptyset$).



- $\{8, 4, 7, 1\} = \{4, 1, 7, 8\}$.
- El orden en que se escriben los elementos es irrelevante, estos conjuntos son indistiguibles al no haber linealidad.

Universidad Andrés Bello

Repaso de Notación de Conjuntos

Sea $A = \{1, 2, 3, 4\}$ y $B = \{5, 1, 7, 4\}$.

- Pertenencia: $x \in A$.
- No pertenencia: $x \notin A$.
- Conjunto vacío: $C = \emptyset$.
- Conjunto universal: $\mathbb{U}$.
- Inclusión: $A \subset B$.
- Intersección: $A \cap B$.
- Unión: $A \cup B$.
- Diferencia: $A - B$.

Universidad Andrés Bello

Operaciones más comunes

Se tienen los conjuntos A, B, C y x como de tipo elemento. Las operaciones básicas serían:

- `C = Union(A, B)` $\rightarrow C = A \cup B$.
- `C = Interseccion(A, B)` $\rightarrow C = A \cap B$.
- `C = Diferencia(A, B)` $\rightarrow C = A - B$.
- `C = Combina(A, B)` $\rightarrow C = A \cup B$, con $A \cap B = \emptyset$ (no se repiten elementos, está supeditada a la implementación).
- `Pertenece(x, A)` $\rightarrow$ verdadero si $x \in A$, falso en caso contrario.
- `Anular(A)` $\rightarrow A = \emptyset$. vacía el conjunto A.
- `Insertar(x, A)` $\rightarrow A = A \cup \{x\}$.
- `Suprime(x, A)` $\rightarrow A = A - \{x\}$.
- `Asigna(A, B)` $\rightarrow A = B$.
- `Max(A)` $\rightarrow$ Devuelve el mayor elemento de A.
- `Min(A)` $\rightarrow$ Devuelve el menor elemento de A.
- `Igual(A, B)` $\rightarrow$ Devuelve Verdadero si $A = B$, falso $A \neq B$.
- `Encuentra(x)` $\rightarrow$ Devuelve el conjunto al que pertenece x.

Universidad Andrés Bello

Operaciones más comunes

Ejemplos: Sea $A = \{1, 2, 3, 4\}$ y $B = \{5, 1, 7, 4\}$.

- `C = Union(A, B).`
- `C = Interseccion(A, B).`
- `C = Diferencia(A, B).`
- `C = Combina(A, B).`
- `Pertenece(5, A).`
- `Anular(A).`
- `Insertar(1, A).`
- `Insertar(5, A).`
- `Suprime(5, A).`
- `Suprime(1, A).`
- `Asigna(A, B).`
- `Max(A).`
- `Min(A).`
- `Igual(A, B).`
- `Encuentra(1).`

Operaciones más comunes

Ejemplos: Sea $A = \{1, 2, 3, 4\}$ y $B = \{5, 1, 7, 4\}$.

- `C = Union(A, B)` $\rightarrow C = \{1, 2, 3, 4, 5, 7\}$.
- `C = Interseccion(A, B)` $\rightarrow C = \{1, 4\}$.
- `C = Diferencia(A, B)` $\rightarrow C = \{2, 3\}$.
- `C = Combinar (A, B)` $\rightarrow$ Error ya que $A \cap B = \emptyset$.
- `Pertenece(5, A)` $\rightarrow$ Falso.
- `Anular(A)` $\rightarrow A = \emptyset$.
- `Insertar(1, A)` $\rightarrow A = \{1, 2, 3, 4\}$.
- `Insertar(5, A)` $\rightarrow A = \{1, 2, 3, 4, 5\}$.
- `Suprime(5, A)` $\rightarrow A = \{1, 2, 3, 4\}$.
- `Suprime(1, A)` $\rightarrow A = \{2, 3, 4\}$.
- `Asigna(A, B)` $\rightarrow A = \{5, 1, 7, 4\}$.
- `Max(A)` $\rightarrow 4$.
- `Min(A)` $\rightarrow 1$.
- `Igual(A, B)` $\rightarrow$ Falso.
- `Encuentra(1)` $\rightarrow$ Error ya que $1 \in A$ y $1 \in B$. `Encuentra(2)` $\rightarrow$ A.

Universidad Andrés Bello

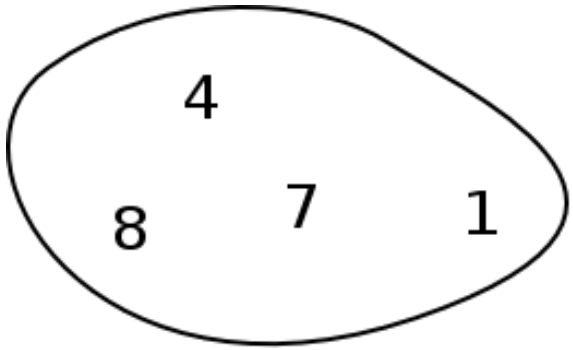
Tipos de Implementaciones de Conjuntos

- ¿Cómo representar el tipo conjunto, de forma que las operaciones se ejecuten rápidamente, con un uso razonable de memoria?
- Tres tipos de implementaciones básicas:
  - Mediante arreglos de booleanos (de *bits*).
  - Mediante arreglos.
  - Mediante listas.
- La mejor implementación depende de cada aplicación concreta:
  - La elección depende de la frecuencia de uso de las operaciones.
  - El Tamaño y la variabilidad de los conjuntos usados.
  - Etc.

Implementación

Arreglos de bits

- Cada elemento se representa por un *bit*, este *bit* toma los valores:
  - 1 si elemento pertenece al conjunto.
  - 0 si elemento no pertenece al conjunto.
- Es por esto que un conjunto se puede representar con algún tipo de dato primitivo.



8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	1	0

Implementación

Arreglos de bits

- Ahora bien, se va a emplear un arreglo de *bits*.
- No se va a emplear un arreglo/*array*/vector como tal.
- Sino un tipo de datos definido por el lenguaje de programación, que suele ocupar entre 8 y 64 *bits*, y por tanto podrá incluir hasta 64 elementos en el conjunto. Por ejemplo, en C se define un tipo que ocupa 8 *bits* :

```
unsigned char set;
```

- Si todos los *bits* de `set` están a 1 entonces se tiene el conjunto: $A = \{0, 1, 2, 3, 4, 5, 6, 7\}$, y su cardinal es 8.
- Si todos los *bits* están a 0 se tiene el conjunto vacío.
- El *bit* más significativo señalará al elemento de mayor valor, el *bit* menos significativo al de menor valor.

Implementación

Arreglos de bits

- Ejemplos (*bit* más significativo a la izquierda):

$$11111111 \rightarrow A = \{0, 1, 2, 3, 4, 5, 6, 7\}$$
$$11110001 \rightarrow A = \{0, 4, 5, 6, 7\}$$
$$01010101 \rightarrow A = \{0, 2, 4, 6\}$$
$$00000000 \rightarrow A = \emptyset$$

- La razón para emplear los arreglo de *bits* es que las operaciones sobre los conjuntos se realizan de manera muy rápida y sencilla.
- Al menos con los computadores actuales, que tienen un tamaño de palabra múltiplo de 8.
- Por supuesto, la ocupación en memoria está optimizada al máximo.
- El inconveniente es que el rango de representación es muy limitado.
- Por eso su aplicación es muy restringida, y depende fuertemente del compilador y el computador sobre el que se implementan, pero es increíblemente rápida.

Implementación de Arreglos de bits

Operaciones

Unión: con el operador `OR` inclusivo binario (|). Ejemplo:

$$\begin{array}{ll} A = \{1,2,5,6,7\} & 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\ B = \{2,3,4,7\} & 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \\ C = \{1,2,3,4,5,6,7\} & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0 \end{array}$$

Intersección: con el operador `AND` binario (&). Ejemplo:

$$\begin{array}{ll} A = \{1,2,5,6,7\} & 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\ B = \{2,3,4,7\} & 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \\ C = \{2,7\} & 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0 \end{array}$$

Implementación de Arreglos de bits

Operaciones

Diferencia: Para obtener $C = A - B$ se invierten todos los *bits* de B y se hace un `AND` entre A y B negado ($\neg$). Ejemplo:

$$\begin{array}{ll} A = \{1,2,5,6,7\} & 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\ B = \{2,3,4,7\} & 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \\ \neg B = \{0,1,5,6\} & 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1 \end{array}$$

Entonces la operación `AND`:

$$\begin{array}{ll} A = \{1,2,5,6,7\} & 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\ \neg B = \{0,1,5,6\} & 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1 \\ C = \{1,5,6\} & 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0 \end{array}$$

- Nota: Complemento a uno o negación en C es $\sim$.

Implementación de Arreglos de bits

Operaciones

Diferencia simétrica: Se realiza mediante la operación de `OR` exclusivo (`XOR`) o $C = (A - B) \cup (B - A)$.

Ejemplo:

XOR (OR exclusivo): Es verdadero, Si una, y solo una de las entradas es verdadera.

Si ambas entradas son falsas o ambas son verdaderas, resulta en una salida falsa.

$$\begin{array}{ll} A = \{1,2,5,6,7\} & 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \\ \neg A = \{0,3,4\} & 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0 \\ B = \{2,3,4,7\} & 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ \neg B = \{0,1,5,6\} & 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0 \end{array}$$

Entonces la operación `AND`:

$$\begin{array}{lll} C = (A - B) & = \{1,\ 5,\ 6\} & 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0 \\ D = (B - A) & = \{3,\ 4\} & 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0 \\ C \cup D & = \{1,\ 3,\ 4,\ 5,\ 6\} & 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0 \end{array}$$

- Nota: `XOR` en C es ^.

Implementación de Arreglos de bits

Operaciones

Igualdad de conjuntos: La implementación es directa, si todos los *bits* de A y B se corresponden entonces son iguales, $A = B$.

Subconjuntos: Si un conjunto A es subconjunto (considerando que un conjunto cualquiera es subconjunto de si mismo) de otro B entonces verifica esta relación: A intersección $B = A$. Notar que A es subconjunto de A, pues A intersección $A = A$.

- Ejemplo: $A = \{1, 2, 3, 4\}$, $B = \{0, 1, 2, 3\}$ $C = A$ intersección $B = \{1, 2, 3\}$; C es distinto de A.

Implementación de Arreglos de bits

Operaciones

Pertenencia: Se requiere efectuar una operación de desplazamiento a nivel de *bits* y una posterior comprobación del *bit* de signo resultante.

- Para esto se utiliza el *bit* de desplazamiento a la derecha $>>$ y realizar el test del primer *bit*.
- Por ejemplo el conjunto:

$$A = 01010110 \text{ (en realidad es 86) corresponde a } \{1, 2, 4, 6\}$$

- Supongamos que queremos verificar si es que se encuentra el 4.
- Entonces, $A >> 4$ da como resultado mover cuatro *bits* hacia la derecha, es como si empujásemos 4 ceros por la izquierda: 00000101.
- Luego sobre el resultado aplicamos módulo 2, si este nos resulta 1, entonces pertenece.
- Pues el resultado de 00000101 % 2 es 00000001.
- En resumen la pertenencia corresponde a: `if`$((A >> x)\,\%2)$

Implementación de Arreglos de bits

Operaciones

Insertar: Para insertar un elemento x, es necesario poner a 1 el *bit* correspondiente. Hay que sumar un valor que se corresponda con el *bit* que se quiere establecer a 1. Aplicando un desplazamiento hacia la izquierda sobre el número 1. Se desplazan x *bits* hacia la izquierda.

- Por ejemplo, partir de A = 0 = conjunto vacío = $\{\ \} = \emptyset$.
- Se quieren insertar los elementos 0,2,3 sobre A.
- Insertar 0: $x = 0$, (00000001 en binario). Se desplaza $A + (1 << 0)$ y se obtiene $A = 1$.
- Insertar 2: $x = 2$. Se desplaza $A + (1 << 2)$ y se obtiene $A = 5$ (000000101).
- Insertar 3: $x = 3$. Se desplaza $A + (1 << 3)$ y se obtiene $A = 13$ (00001101).
- Para `Borrar` es exactamente lo mismo, pero hay que restar en vez de sumar.
- Ejemplo: borrar 3 de A. Se desplaza $A - (1 << 3)$ y se obtiene $A = 5$ (00000101).

Implementación de Arreglos de bits

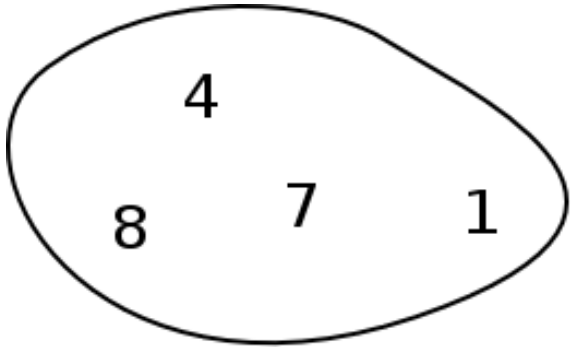
Ventajas y Desventajas

- La gran ventaja de esta implementación es la rapidez de ejecución de todas las operaciones, que se ejecutan en tiempo constante: $O(1)$.
- Además los elementos se encuentran empaquetados ocupando el menor espacio posible, esto es, un único bit.
- La desventaja es que no admiten un rango muy amplio de representación.
- Aun así, para incrementar el rango basta con crear un arreglo de tipo conjunto, por ejemplo: `unsigned char superconjunto[10]`, y aplicar las operaciones sobre los *bits* en todos los elementos del arreglo, excepto para la inserción y borrado, en cuyo caso hay que encontrar el *bit* exacto a manipular.

Universidad Andrés Bello

Implementación con Arreglos

- Sea $A = \{1, 4, 7, 8\}$.
- Se utiliza un arreglo de enteros y se van ingresando los elementos.
- Los elementos dentro del arreglo no están ordenados entre sí.
- En el ejemplo la Cardinalidad es 4



0	1	2	3	4	5	6	7	8
1	4	7	8	0	0	0	0	0

Implementación con Arreglos

- Esta representación no limita el rango de representación más que al tipo de datos empleado.
- Por razones de eficiencia a la hora de implementar las primitivas, las estructuras se pasan por referencia.
- No se implementan rutinas de control de errores ni su detección. Se produce un error cuando se tratan de añadir elementos y estos desbordan la capacidad del arreglo.

Implementación con Arreglos

- Tipo de datos empleado:

```
typedef struct conjunto{
  int elemetos[MAXELEM];
  int cardinal;
} Conjunto;
```

- Definición de conjunto vacío:
  - Un conjunto está vacío si su cardinal es cero. Para inicializar un conjunto a vacío basta con una instrucción: `A->cardinal = 0`

Universidad Andrés Bello

Implementación con Arreglos

Primero estos operadores porque los demás operadores los usan.

Pertenece: Para determinar si un elemento x pertenece al conjunto se recorre el arreglo hasta encontrarlo. Se devuelve verdadero si se encuentra.

Insertar: Primero se debe comprobar que no está el elemento, después se inserta en la última posición, esto es, la que señale el cardinal, que se incrementa en una unidad.

Borrar: No se puede eliminar el elemento y dejar un espacio, puesto que en ese caso ya no se tiene una lista. Para eliminar este problema se sustituye el elemento borrado por el último de la lista.

Universidad Andrés Bello

Implementación con Arreglos

Unión: Para $C = A \cup B$, se introducen en C todos los elementos de A y todos los elementos de B que no pertenezcan a A.

Intersección: Para $C = A \cap B$, se hace un recorrido sobre A (o B) y se insertan en C los elementos que estén en B (o A). Es decir:

Algorithm 1 `Interseccion`

1: $C \leftarrow \emptyset$
2: **for** cada x elemento de A **do**
3: si x pertenece a B entonces insertar x en C
4: **end for**

Implementación con Arreglos

Diferencia: Para hacer $C = A - B$, se hace un recorrido sobre A (o B) y se insertan en C los elementos que no estén en B (o A). Es decir

Algorithm 2 `Diferencia`

1: $C \leftarrow \emptyset$
2: **for** cada x elemento de A **do**
3: si x no pertenece a B entonces insertar x en C
4: **end for**

Universidad Andrés Bello

Implementación con Arreglos

Diferencia simétrica: Sea $C = (A - B) \cup (B - A)$. Para obtener este resultado se replica lo anterior (diferencia y unión).

Algorithm 3 `Diferencia`

1: $C \leftarrow \emptyset$
2: **for** cada x elemento de A **do**
3: si x no pertenece a B entonces insertar x en C
4: **end for**
5: **for** cada x elemento de B **do**
6: si x no pertenece a A entonces insertar x en C
7: **end for**

Universidad Andrés Bello

Implementación con Arreglos

Subconjuntos: Se debe comprobar si todo elemento de A es elemento de B. Se devuelve verdadero si A es subconjunto de B.

Igualdad de conjuntos: Un conjunto A es igual a otro B si A es subconjunto de B y ambos tienen los mismos elementos. Se devuelve verdadero si A es igual a B.

Implementación con Arreglos

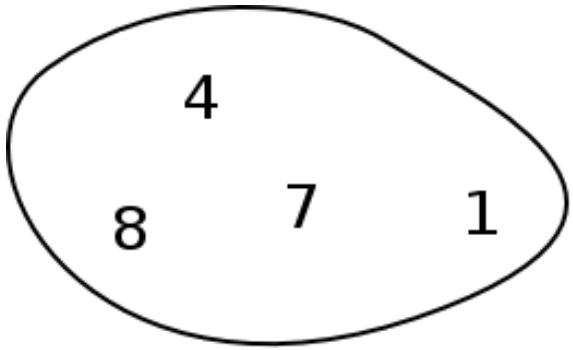
Ventajas y Desventajas

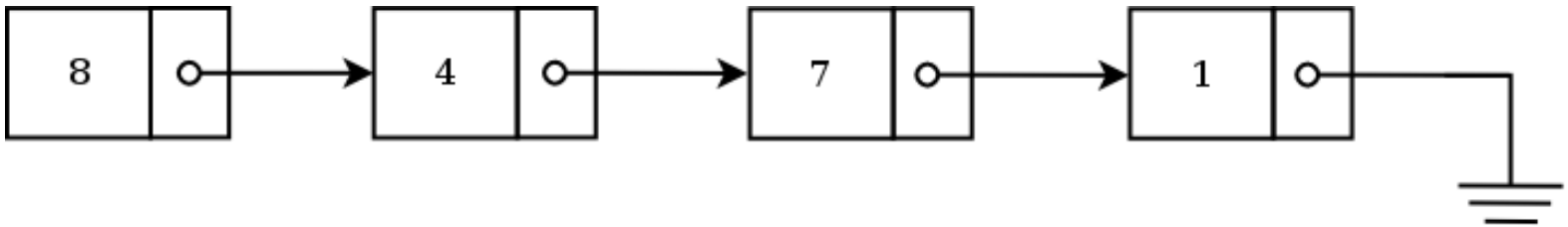
- La ventaja de esta implementación es que no limita el rango de representación de los elementos del conjunto.
- Tampoco limita el tipo de datos, siempre y cuando se pueda deducir cuando un elemento es igual a otro o no.
- La desventaja de esta implementación con respecto a la de arreglos de bits es su mala eficacia con respecto al tiempo de ejecución.
- El coste de la inserción y borrado es $O(1)$.
- Siendo $|A|$ el cardinal de un conjunto cualquiera A las operaciones de pertenencia se ejecuta en un tiempo $O(|A|)$.
- En las restantes operaciones, que implican a dos conjuntos, la complejidad es $O(|A| \cdot |B|)$.
- El espacio que ocupa un conjunto es de $O(n)$, siendo n el tamaño del arreglo.

Universidad Andrés Bello

Implementación con Listas

- Guardar en una lista los elementos que pertenecen al conjunto





Universidad Andrés Bello

Implementación con Listas

Ventajas:

- Utiliza espacio proporcional al tamaño del conjunto representado (no al conjunto universal).
- El conjunto universal puede ser muy grande, o incluso infinito.

Inconvenientes:

- Las operaciones son menos eficientes si el conjunto universal es reducido.
- Gasta más memoria y tiempo si los conjuntos están muy llenos.
- Más complejo de implementar.

Implementación con Listas

Algorithm 4 Pertenece(x, A)

1: $p = A \rightarrow inicio$
2: **while** $p \rightarrow dato \neq x$ y $p \neq NULL$ **do**
3: $p = p \rightarrow sig$
4: **end while**
5: **if** $p \neq NULL$ **then**
6: retornar verdadero
7: **else**
8: retornar falso
9: **end if**

Algorithm 5 Interseccion(A, B)

1: $C = \emptyset$
2: $p = A \rightarrow inicio$
3: **while** $p \neq NULL$ **do**
4: **if** $Pertenece(p \rightarrow dato, A)$ **then**
5: Insertar(C, $p \rightarrow dato$)
6: **end if**
7: $p = p \rightarrow sig$
8: **end while**
9: retornar C

Universidad Andrés Bello

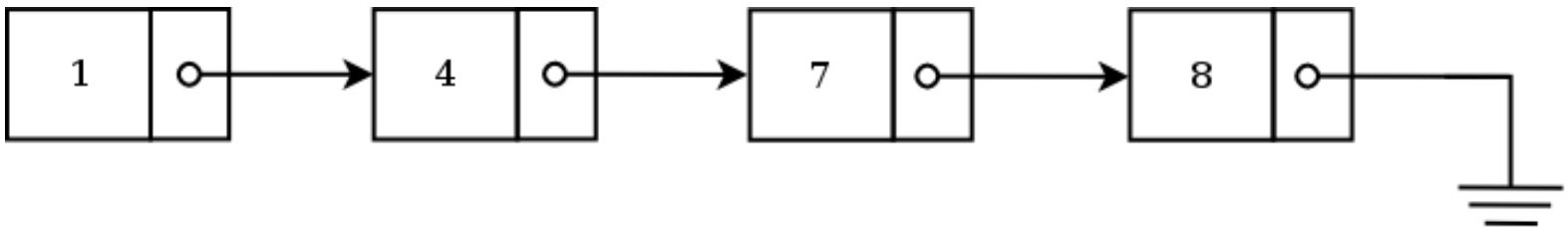
Implementación con Listas

- ¿Cuánto tiempo tardan las operaciones anteriores?
- Suponemos una lista de tamaño n y otra m (o ambas de tamaño n).
- ¿Cómo sería Unión, Diferencia, Insertar, Borrar, etc.?
- Inconveniente: Unión, Intersección y Diferencia recorren la lista B muchas veces (una por cada elemento de A).
- Se puede mejorar usando listas ordenadas.

Universidad Andrés Bello

Implementación con Listas

- Si ordenamos:



- **Pertenece, Insertar, Borrar:** Parar si encontramos un elemento mayor que el buscado.
- **Unión, Intersección, Diferencia**: Recorrido simultáneo (y único) de ambas listas.

Implementación con Listas

Algorithm 6 `Pertenece(x, A)`

```
1: p = A → inicio
2: while p → dato ≠ x y p ≠ NULL do
3:     p = p → sig
4: end while
5: if p ≠ NULL then
6:     retornar verdadero
7: else
8:     retornar falso
9: end if
```

- ¿Cuánto es el tiempo de ejecución ahora?

Universidad Andrés Bello

Implementación con Listas

Algorithm 7 `UNION(A, B)`

1: $C = \emptyset$
2: $p_a = A \rightarrow inicio$
3: $p_b = B \rightarrow inicio$
4: **while** $p_a \neq NULL$ y $p_b \neq NULL$ **do**
5: **if** $p_b \neq NULL$ o $p_a \rightarrow dato < p_b \rightarrow dato$ **then**
6: Insertar(C, $p_a \rightarrow dato$)
7: $p_a = p \rightarrow sig$
8: **else if** $p_a \neq NULL$ o $p_b \rightarrow dato < p_a \rightarrow dato$ **then**
9: Insertar(C, $p_b \rightarrow dato$)
10: $p_b = p \rightarrow sig$
11: **else**
12: Insertar(C, $p_a \rightarrow dato$)
13: $p_a = p \rightarrow sig$
14: $p_b = p \rightarrow sig$
15: **end if**
16: **end while**
17: retornar C

Implementación con Listas

- ¿Cuánto es el tiempo de ejecución? ¿Es sustancial la mejora?
- ¿Cómo serían la Intersección y la Diferencia?
- ¿Cómo serían las operaciones Min, Max?
- ¿Cuánto es el uso de memoria para tamaño n? Supongamos que 1 puntero $= k_1$ *bytes*, 1 elemento $= k2$ *bytes*.

Universidad Andrés Bello

Implementación con Listas

- Esta implementación tampoco limita el rango de representación de los elementos del conjunto, y por supuesto tampoco limita el tipo de datos, siempre y cuando se pueda deducir cuando un elemento es igual a otro o no.
- Dado un conjunto A y B, las operaciones de inserción, borrado y pertenencia se ejecutan en un tiempo de $O(|A|)$. Las operaciones de unión, intersección, diferencia, diferencia simétrica, subconjunto e igualdad se ejecutan en un tiempo de $O(|A| + |B|)$.
- El espacio que ocupa un conjunto es de $O(|A|)$, siendo $|A|$ el cardinal del conjunto A.
- Por supuesto es proporcional al tamaño del conjunto implementado mediante arreglo, multiplicado por una constante debido al espacio ocupado por los punteros.

Conclusiones Finales

- Arreglos de booleanos: muy rápida para las operaciones de inserción y consulta. Inviable si el tamaño del conjunto universal es muy grande.
- Listas de elementos: uso razonable de memoria, proporcional al tamaño usado. Muy ineficiente para la inserción y consulta de un elemento.
- Solución: Tablas de dispersión (tablas de *hash*), estructuras de árbol, combinación de estructuras, etc.